A sound unification algorithm based on telescope equivalences

Jesper Cockx

DistriNet – KU Leuven

20 April 2016

Pattern matching is awesome

Agda uses unification to:

- check which constructors are possible
- specialize the result type

$$
\begin{aligned}
&\textbf{data } \texttt{Vec}\ (A : \texttt{Set}) : \mathbb{N} \to \texttt{Set} \textbf{ where}\\
&\quad [] : \texttt{Vec}\ A\ 0\\
&\quad \texttt{cons} : (n : \mathbb{N}) \to A \to \texttt{Vec}\ A\ n\\
&\qquad\qquad \to \texttt{Vec}\ A\ (1 + n)
\end{aligned}
$$

$$
\begin{aligned}
&f : \texttt{Vec}\ A\ 1 \to T\\
&f\ (\texttt{cons}\ .0\ x\ xs) = \ldots
\end{aligned}
$$

Pattern matching is awesome

Agda uses unification to:

- check which constructors are possible
- specialize the result type

data $\texttt{Vec}\ (A : \texttt{Set}) : \mathbb{N} \to \texttt{Set}$ **where**

$\quad [] : \texttt{Vec}\ A\ 0$

$\quad \texttt{cons} : (n : \mathbb{N}) \to A \to \texttt{Vec}\ A\ n$

$\qquad\quad \to \texttt{Vec}\ A\ (1 + n)$

$f : \texttt{Vec}\ A\ 1 \to T$

$f\ (\texttt{cons}\ .0\ x\ xs) = \ldots$

Pattern matching is awesome

Agda uses unification to:

- check which constructors are possible
- specialize the result type

$$\textbf{data}\ \texttt{Vec}\ (A : \texttt{Set}) : \mathbb{N} \to \texttt{Set}\ \textbf{where}$$
$$\quad [] : \texttt{Vec}\ A\ 0$$
$$\quad \texttt{cons} : (n : \mathbb{N}) \to A \to \texttt{Vec}\ A\ n$$
$$\qquad\qquad \to \texttt{Vec}\ A\ (1 + n)$$

$$f : \texttt{Vec}\ A\ 1 \to T$$
$$f\ (\texttt{cons}\ .0\ x\ xs) = \ldots$$

Details of unification are important

Agda has pattern matching as a primitive,
so results of unification determine
Agda's notion of equality

Example: deleting reflexive equations implies K

Details of unification are important

Agda has pattern matching as a primitive,
so results of unification determine
Agda's notion of equality

Example: deleting reflexive equations implies K

Time for a quiz

Should the following code be accepted?

```
{-# OPTIONS --without-K #-}
... -- imports

f : (Bool , true) ≡ (Bool , false) → ⊥
f ()
```

Time for a quiz

Should the following code be accepted?

```
{-# OPTIONS --without-K #-}
... -- imports

f : (Bool , true) ≡ (Bool , false) → ⊥
f ()
```

Answer: depends on the type of the equation!

Postponing equations causes problems

If we postpone an equation,
following equations can be heterogeneous

Naively continuing unification is bad

- Equality of second projections
- Injectivity of type constructors
- ...

It's hard to distinguish good and bad situations!

Postponing equations causes problems

If we postpone an equation,
following equations can be heterogeneous

Naively continuing unification is bad

- Equality of second projections
- Injectivity of type constructors
- ...

It's hard to distinguish good and bad situations!

Postponing equations causes problems

If we postpone an equation,
following equations can be heterogeneous

Naively continuing unification is bad

- Equality of second projections
- Injectivity of type constructors
- ...

It's hard to distinguish good and bad situations!

We need a general way to think about unification

It's not sufficient to "make things equal"

Core idea:

> *Unification rules are equivalences between telescopes of equations*

This is the basis of the new unification algorithm in Agda 2.5.1

We need a general way to think about unification

It's not sufficient to “make things equal”

Core idea:

> *Unification rules are equivalences between telescopes of equations*

This is the basis of the new unification algorithm in Agda 2.5.1

We need a general way to think about unification

It's not sufficient to "make things equal"

Core idea:

> *Unification rules are equivalences*
> *between telescopes of equations*

This is the basis of the new unification algorithm in Agda 2.5.1

A sound unification algorithm based on telescope equivalences

1. Unifiers as equivalences
2. Unification rules
3. Higher-dimensional unification

A sound unification algorithm based on telescope equivalences

1. Unifiers as equivalences
2. Unification rules
3. Higher-dimensional unification

What do we want from unification?

It has to be possible to translate pattern matching to eliminators

The core tool we need is **specialization by unification**

Build a function $m : \Gamma \to \bar{u} \equiv_{\Delta} \bar{v} \to T$ from a function $m' : \Gamma' \to T\sigma$ where $\sigma : \Gamma' \to \Gamma$ is computed by unification

What do we want from unification?

It has to be possible to translate pattern matching to eliminators

The core tool we need is **specialization by unification**

Build a function $m : \Gamma \to \bar{u} \equiv_{\Delta} \bar{v} \to T$
from a function $m' : \Gamma' \to T\sigma$
where $\sigma : \Gamma' \to \Gamma$ is computed by unification

Intermezzo: telescopic equality

Type of an equation may depend
on solution of previous equations

Heterogeneous equality doesn't
keep enough information:

- Safe to consider equation homogeneous?
- Does equation depend on other equation?
- How do equations depend on each other?

Intermezzo: telescopic equality

Type of an equation may depend
on solution of previous equations

Heterogeneous equality doesn't
keep enough information:

- Safe to consider equation homogeneous?
- Does equation depend on other equation?
- How do equations depend on each other?

Intermezzo: telescopic equality

Solution: use "path over" construction
to keep track of dependencies

For example:

$$(e_1 : m \equiv_{\mathbb{N}} n)(e_2 : u \equiv^{e_1}_{\texttt{Vec}\ A} v)$$

Cubical (abuse of) notation:

$$(e_1 : m \equiv_{\mathbb{N}} n)(e_2 : u \equiv_{\texttt{Vec}\ A\ e_1} v)$$

Intermezzo: telescopic equality

Solution: use "path over" construction
to keep track of dependencies

For example:

$$(e_1 : m \equiv_{\mathbb{N}} n)(e_2 : u \equiv^{e_1}_{\texttt{Vec}\ A} v)$$

Cubical (abuse of) notation:

$$(e_1 : m \equiv_{\mathbb{N}} n)(e_2 : u \equiv_{\texttt{Vec}\ A\ e_1} v)$$

Specialization by unification

The goal is to construct $m : \Gamma \to \bar{u} \equiv_{\Delta} \bar{v} \to T$

Input:

- Telescope Γ of *flexible variables*
- Telescope $\bar{u} \equiv_{\Delta} \bar{v}$ of equations

Output:

- New telescope Γ'
- Substitution $\sigma : \Gamma' \to \Gamma$
- Evidence of unification $\bar{e} : \Gamma' \to \bar{u}\sigma \equiv_{\Delta\sigma} \bar{v}\sigma$

Specialization by unification

The goal is to construct $m : \Gamma \to \bar{u} \equiv_\Delta \bar{v} \to T$

Input:

- Telescope Γ of *flexible variables*
- Telescope $\bar{u} \equiv_\Delta \bar{v}$ of equations

Output:

- New telescope Γ'
- Substitution $\sigma : \Gamma' \to \Gamma$
- Evidence of unification
  $\bar{e} : \Gamma' \to \bar{u}\sigma \equiv_{\Delta\sigma} \bar{v}\sigma$

Specialization by unification

The goal is to construct $m : \Gamma \to \bar{u} \equiv_{\Delta} \bar{v} \to T$

Input:

- Telescope Γ of *flexible variables*
- Telescope $\bar{u} \equiv_{\Delta} \bar{v}$ of equations

Output:

- New telescope Γ'
- Substitution $\sigma : \Gamma' \to \Gamma$
- Evidence of unification
  $\bar{e} : \Gamma' \to \bar{u}\sigma \equiv_{\Delta\sigma} \bar{v}\sigma$

Specialization by unification

The goal is to construct $m : \Gamma \to \bar{u} \equiv_\Delta \bar{v} \to T$

Input:

- Telescope Γ of *flexible variables*
- Telescope $\bar{u} \equiv_\Delta \bar{v}$ of equations

Output:

- New telescope Γ'
- Telescope mapping $f : \Gamma' \to \Gamma(\bar{u} \equiv_\Delta \bar{v})$

Two more requirements

Let $f : \Gamma' \to \Gamma(\bar{u} \equiv_\Delta \bar{v})$ be a unifier

- f should be most general
  $\Rightarrow$ f needs a *right inverse* g_1

- Γ' should be minimal
  $\Rightarrow$ f needs a *left inverse* g_2

Two more requirements

Let $f : \Gamma' \to \Gamma(\bar{u} \equiv_\Delta \bar{v})$ be a unifier

- f should be most general
  $\Rightarrow f$ needs a *right inverse* g_1
- Γ' should be minimal
  $\Rightarrow f$ needs a *left inverse* g_2

Two more requirements

Let $f : \Gamma' \to \Gamma(\bar{u} \equiv_\Delta \bar{v})$ be a unifier

- f should be most general
  $\Rightarrow$ f needs a *right inverse* g_1
- Γ' should be minimal
  $\Rightarrow$ f needs a *left inverse* g_2

Most general unifiers as equivalences

A most general unifier of $\bar{u}$ and $\bar{v}$ is an equivalence $f : \Gamma(\bar{u} \equiv_\Delta \bar{v}) \simeq \Gamma'$ for some Γ'

Specialization by unification:

$$m : \Gamma \to \bar{u} \equiv_\Delta \bar{v} \to T$$
$$m\ \bar{x}\ \bar{e} = \overline{\texttt{subst}}\ (\lambda \bar{x}\ \bar{e}.\ T)\ (\texttt{isLinv}\ f\ \bar{x}\ \bar{e})$$
$$(m'\ (f\ \bar{x}\ \bar{e}))$$

Most general unifiers as equivalences

A most general unifier of $\bar{u}$ and $\bar{v}$ is an equivalence $f : \Gamma(\bar{u} \equiv_\Delta \bar{v}) \simeq \Gamma'$ for some Γ'

Specialization by unification:

$$
\begin{aligned}
& m : \Gamma \to \bar{u} \equiv_\Delta \bar{v} \to T \\
& m\ \bar{x}\ \bar{e} = \overline{\mathtt{subst}}\ (\lambda \bar{x}\ \bar{e}.\ T)\ (\mathtt{isLinv}\ f\ \bar{x}\ \bar{e}) \\
& \qquad (m'\ (f\ \bar{x}\ \bar{e}))
\end{aligned}
$$

Disunifiers

A disunifier of $\bar{u}$ and $\bar{v}$ is an equivalence
$f : \Gamma(\bar{u} \equiv_\Delta \bar{v}) \simeq \bot$

Specialization by unification:

$$m : \Gamma \to \bar{u} \equiv_\Delta \bar{v} \to T$$
$$m\ \bar{x}\ \bar{e} = \mathtt{elim}_\bot\ T\ (f\ \bar{x}\ \bar{e})$$

Disunifiers

A disunifier of $\bar{u}$ and $\bar{v}$ is an equivalence $f : \Gamma(\bar{u} \equiv_\Delta \bar{v}) \simeq \bot$

Specialization by unification:

$$m : \Gamma \to \bar{u} \equiv_\Delta \bar{v} \to T$$
$$m\ \bar{x}\ \bar{e} = \mathtt{elim}_\bot\ T\ (f\ \bar{x}\ \bar{e})$$

A sound unification algorithm based on telescope equivalences

Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

$$
\begin{aligned}
&(k\ l : \mathbb{N})(\underline{e} : \mathtt{suc}\ k \equiv_{\mathbb{N}} \mathtt{suc}\ l) \\
&\simeq (k\ \underline{l} : \mathbb{N})(\underline{e} : k \equiv_{\mathbb{N}} l) \\
&\simeq (k : \mathbb{N})
\end{aligned}
$$

$$f^{-1} : (k : \mathbb{N}) \to (k\ l : \mathbb{N})(e : \mathtt{suc}\ k \equiv_{\mathbb{N}} \mathtt{suc}\ l)$$
$$f^{-1}\ k = k; k; \mathtt{refl}$$

Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

$$
\begin{aligned}
&(k\ l : \mathbb{N})(\underline{e} : \mathtt{suc}\ k \equiv_{\mathbb{N}} \mathtt{suc}\ l) \\
&\quad \simeq (k\ \underline{l} : \mathbb{N})(\underline{e} : k \equiv_{\mathbb{N}} l) \\
&\quad \simeq (k : \mathbb{N})
\end{aligned}
$$

$$f^{-1} : (k : \mathbb{N}) \to (k\ l : \mathbb{N})(e : \mathtt{suc}\ k \equiv_{\mathbb{N}} \mathtt{suc}\ l)$$

$$f^{-1}\ k = k; k; \mathtt{refl}$$

Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

$$
\begin{aligned}
&(k\ l : \mathbb{N})(\underline{e} : \mathtt{suc}\ k \equiv_{\mathbb{N}} \mathtt{suc}\ l) \\
&\simeq (k\ \underline{l} : \mathbb{N})(\underline{e} : k \equiv_{\mathbb{N}} l) \\
&\simeq (k : \mathbb{N})
\end{aligned}
$$

$f^{-1} : (k : \mathbb{N}) \to (k\ l : \mathbb{N})(e : \mathtt{suc}\ k \equiv_{\mathbb{N}} \mathtt{suc}\ l)$

$f^{-1}\ k = k; k; \mathtt{refl}$

Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

$$
\begin{aligned}
&(k\ l : \mathbb{N})(\underline{e} : \mathtt{suc}\ k \equiv_{\mathbb{N}} \mathtt{suc}\ l) \\
&\ \simeq (k\ \underline{l} : \mathbb{N})(\underline{e} : k \equiv_{\mathbb{N}} l) \\
&\ \simeq (k : \mathbb{N})
\end{aligned}
$$

$f^{-1} : (k : \mathbb{N}) \to (k\ l : \mathbb{N})(e : \mathtt{suc}\ k \equiv_{\mathbb{N}} \mathtt{suc}\ l)$
$f^{-1}\ k = k; k; \mathtt{refl}$

Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

$$\begin{aligned} &(k\ l : \mathbb{N})(\underline{e} : \texttt{suc}\ k \equiv_{\mathbb{N}} \texttt{suc}\ l) \\ &\ \underline{\simeq}\ (k\ \underline{l} : \mathbb{N})(\underline{e} : k \equiv_{\mathbb{N}} l) \\ &\ \underline{\simeq}\ (k : \mathbb{N}) \end{aligned}$$

$f^{-1} : (k : \mathbb{N}) \to (k\ l : \mathbb{N})(e : \texttt{suc}\ k \equiv_{\mathbb{N}} \texttt{suc}\ l)$
$f^{-1}\ k = k; k; \texttt{refl}$

Basic unification rules

$$
\begin{aligned}
\text{Solution:}\quad & (x : A)(e : x \equiv_A t) \simeq () \\
\text{Deletion:}\quad & (f\ x \equiv_{\mathbb{N}} f\ x) \simeq () \\
\text{Injectivity:}\quad & (\mathtt{suc}\ x \equiv_{\mathbb{N}} \mathtt{suc}\ y) \simeq (x \equiv_{\mathbb{N}} y) \\
\text{Conflict:}\quad & (\mathtt{inj}_1\ x \equiv_{A \uplus B} \mathtt{inj}_2\ y) \simeq \bot \\
\text{Cycle:}\quad & (n \equiv_{\mathbb{N}} \mathtt{suc}\ n) \simeq \bot
\end{aligned}
$$

$+$ auxiliary rules for weakening and reordering

Rules for η-equality of records

η-expansion of a flexible variable:

$$(\underline{p} : \mathbb{N} \times \mathbb{N})(e : \mathtt{fst}\ p \equiv_{\mathbb{N}} \mathtt{zero})$$
$$\simeq (\underline{x} : \mathbb{N})(y : \mathbb{N})(\underline{e} : x \equiv_{\mathbb{N}} \mathtt{zero})$$
$$\simeq (y : \mathbb{N})$$

η-expansion of an equation:

$$(e : x, y \equiv_{\mathbb{N}\times\mathbb{N}} f\ z)$$
$$\simeq \begin{array}{l}(e_1 : x \equiv_{\mathbb{N}} \mathtt{fst}\ (f\ z)) \\ (e_2 : y \equiv_{\mathbb{N}} \mathtt{snd}\ (f\ z))\end{array}$$

Rules for η-equality of records

η-expansion of a flexible variable:

$$
\begin{aligned}
&(\underline{p} : \mathbb{N} \times \mathbb{N})(e : \texttt{fst}\ p \equiv_{\mathbb{N}} \texttt{zero}) \\
&\simeq (\underline{x} : \mathbb{N})(y : \mathbb{N})(\underline{e} : x \equiv_{\mathbb{N}} \texttt{zero}) \\
&\simeq (y : \mathbb{N})
\end{aligned}
$$

η-expansion of an equation:

$$
\begin{aligned}
&(e : x, y \equiv_{\mathbb{N} \times \mathbb{N}} f\ z) \\
&\simeq \begin{matrix} (e_1 : x \equiv_{\mathbb{N}} \texttt{fst}\ (f\ z)) \\ (e_2 : y \equiv_{\mathbb{N}} \texttt{snd}\ (f\ z)) \end{matrix}
\end{aligned}
$$

Rules for indexed data types

Idea: rules solve equations between indices together with equations between constructors

Example:

$$(e_1 : \mathtt{suc}\ m \equiv_{\mathbb{N}} \mathtt{suc}\ n)$$
$$(e_2 : \mathtt{cons}\ m\ x\ xs \equiv_{\mathtt{Vec}\ A\ e_1} \mathtt{cons}\ n\ y\ ys)$$
$$\simeq \begin{array}{l}(e_1 : m \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)\\(e_3 : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys)\end{array}$$

Rules for indexed data types

Idea: rules solve equations between indices together with equations between constructors

Example:

$$
\begin{array}{l}
(e_1 : \texttt{suc}\ m \equiv_{\mathbb{N}} \texttt{suc}\ n) \\
(e_2 : \texttt{cons}\ m\ x\ xs \equiv_{\texttt{Vec}\ A\ e_1} \texttt{cons}\ n\ y\ ys)
\end{array}
\simeq
\begin{array}{l}
(e_1 : m \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y) \\
(e_3 : xs \equiv_{\texttt{Vec}\ A\ e_1} ys)
\end{array}
$$

Rules for indexed data types

This can give a real boost to power:

$$\textbf{data}\ \texttt{Im}\ (f : A \to B) : B \to \texttt{Set}\ \textbf{where}$$
$$\quad \texttt{image} : (x : A) \to \texttt{Im}\ f\ (f\ x)$$

$$\begin{aligned}
&(x\ y : A)(\underline{e_1} : f\ x \equiv_B f\ y) \\
&(\underline{e_2} : \texttt{image}\ x \equiv_{\texttt{Im}\ f\ e_1} \texttt{image}\ y) \\
\simeq\ &(x\ y : A)(e : x \equiv_A y) \\
\simeq\ &(x : A)
\end{aligned}$$

From this point, there be dragons

Any questions so far?

A sound unification algorithm based on telescope equivalences

1. Unifiers as equivalences

2. Unification rules

3. Higher-dimensional unification

Indexed rules are too restrictive

Rules for indexed datatypes require indices to be fully general

This is too restrictive:

$$(e_1 : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ n)} \mathtt{cons}\ n\ y\ ys)$$
$$\not\simeq (e_1 : x \equiv_A y)(e_2 : xs \equiv_{\mathtt{Vec}\ A\ n}\ ys)$$

Indexed rules are too restrictive

Rules for indexed datatypes require indices to be fully general

This is too restrictive:

$$(e_1 : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ n)} \mathtt{cons}\ n\ y\ ys)$$
$$\not\simeq (e_1 : x \equiv_A y)(e_2 : xs \equiv_{\mathtt{Vec}\ A\ n} ys)$$

Generalized rules for indexed data

The following rules can be generalized to arbitrary indices:

- Conflict
- Cycle
- Injectivity: only if index types satisfy K!

Reverse unification rules

Idea: we can generalize the indices
by applying unification rules in reverse

Reverse unification rules: example

$(\underline{n} : \mathbb{N})(x\ y : A)(xs\ ys : \mathtt{Vec}\ A\ n)$
$(e : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ n)} \mathtt{cons}\ n\ y\ ys)$

$$\begin{aligned}
& (m\ n : \mathbb{N})(x\ y : A)(xs : \mathtt{Vec}\ A\ m)(ys : \mathtt{Vec}\ A\ n) \\
\simeq\ & (\underline{e_1} : m \equiv_{\mathbb{N}} n) \\
& (e_2 : \mathtt{cons}\ m\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ e_1)} \mathtt{cons}\ n\ y\ ys) \\
& (m\ n : \mathbb{N})(x\ y : A)(xs : \mathtt{Vec}\ A\ m)(ys : \mathtt{Vec}\ A\ n) \\
\simeq\ & (e_1 : \mathtt{suc}\ m \equiv_{\mathbb{N}} \mathtt{suc}\ n) \\
& (\underline{e_2} : \mathtt{cons}\ m\ x\ xs \equiv_{\mathtt{Vec}\ A\ e_1} \mathtt{cons}\ n\ y\ ys) \\
\simeq\ & (\underline{m}\ n : \mathbb{N})(x\ \underline{y} : A)(xs : \mathtt{Vec}\ A\ m)(\underline{ys} : \mathtt{Vec}\ A\ n) \\
& (\underline{e_1} : m \equiv_{\mathbb{N}} n)(\underline{e_2} : x \equiv_A y)(\underline{e_3} : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys) \\
\simeq\ & (n : \mathbb{N})(x : A)(xs : \mathtt{Vec}\ A\ n)
\end{aligned}$$

Reverse unification rules: example

$(\underline{n} : \mathbb{N})(x\ y : A)(xs\ ys : \mathtt{Vec}\ A\ n)$
$(e : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ n)} \mathtt{cons}\ n\ y\ ys)$

$$\begin{aligned}
& (m\ n : \mathbb{N})(x\ y : A)(xs : \mathtt{Vec}\ A\ m)(ys : \mathtt{Vec}\ A\ n) \\
\simeq\ & (\underline{e_1} : m \equiv_{\mathbb{N}} n) \\
& (e_2 : \mathtt{cons}\ m\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ e_1)} \mathtt{cons}\ n\ y\ ys) \\
& (m\ n : \mathbb{N})(x\ y : A)(xs : \mathtt{Vec}\ A\ m)(ys : \mathtt{Vec}\ A\ n) \\
\simeq\ & (e_1 : \mathtt{suc}\ m \equiv_{\mathbb{N}} \mathtt{suc}\ n) \\
& (\underline{e_2} : \mathtt{cons}\ m\ x\ xs \equiv_{\mathtt{Vec}\ A\ e_1} \mathtt{cons}\ n\ y\ ys) \\
\simeq\ & (\underline{m}\ n : \mathbb{N})(x\ \underline{y} : A)(xs : \mathtt{Vec}\ A\ m)(\underline{ys} : \mathtt{Vec}\ A\ n) \\
& (\underline{e_1} : m \equiv_{\mathbb{N}} n)(\underline{e_2} : x \equiv_A y)(\underline{e_3} : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys) \\
\simeq\ & (n : \mathbb{N})(x : A)(xs : \mathtt{Vec}\ A\ n)
\end{aligned}$$

Reverse unification rules: example

$$(\underline{n} : \mathbb{N})(x\ y : A)(xs\ ys : \mathtt{Vec}\ A\ n)$$
$$(e : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ n)} \mathtt{cons}\ n\ y\ ys)$$

$$\simeq \begin{array}{l}(m\ n : \mathbb{N})(x\ y : A)(xs : \mathtt{Vec}\ A\ m)(ys : \mathtt{Vec}\ A\ n)\\ (\underline{e_1} : m \equiv_{\mathbb{N}} n)\\ (e_2 : \mathtt{cons}\ m\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ e_1)} \mathtt{cons}\ n\ y\ ys)\end{array}$$

$$\simeq \begin{array}{l}(m\ n : \mathbb{N})(x\ y : A)(xs : \mathtt{Vec}\ A\ m)(ys : \mathtt{Vec}\ A\ n)\\ (e_1 : \mathtt{suc}\ m \equiv_{\mathbb{N}} \mathtt{suc}\ n)\\ (\underline{e_2} : \mathtt{cons}\ m\ x\ xs \equiv_{\mathtt{Vec}\ A\ e_1} \mathtt{cons}\ n\ y\ ys)\end{array}$$

$$\simeq \begin{array}{l}(\underline{m}\ n : \mathbb{N})(x\ \underline{y} : A)(xs : \mathtt{Vec}\ A\ m)(\underline{ys} : \mathtt{Vec}\ A\ n)\\ (\underline{e_1} : m \equiv_{\mathbb{N}} n)(\underline{e_2} : x \equiv_A y)(\underline{e_3} : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys)\end{array}$$

$$\simeq (n : \mathbb{N})(x : A)(xs : \mathtt{Vec}\ A\ n)$$

Reverse unification rules: example

$(\underline{n} : \mathbb{N})(x\ y : A)(xs\ ys : \mathtt{Vec}\ A\ n)$
$(e : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ n)} \mathtt{cons}\ n\ y\ ys)$

$$\simeq \begin{array}{l}(m\ n : \mathbb{N})(x\ y : A)(xs : \mathtt{Vec}\ A\ m)(ys : \mathtt{Vec}\ A\ n) \\ (\underline{e_1} : m \equiv_{\mathbb{N}} n) \\ (e_2 : \mathtt{cons}\ m\ x\ xs \equiv_{\mathtt{Vec}\ A\ (\mathtt{suc}\ e_1)} \mathtt{cons}\ n\ y\ ys)\end{array}$$

$$\simeq \begin{array}{l}(m\ n : \mathbb{N})(x\ y : A)(xs : \mathtt{Vec}\ A\ m)(ys : \mathtt{Vec}\ A\ n) \\ (e_1 : \mathtt{suc}\ m \equiv_{\mathbb{N}} \mathtt{suc}\ n) \\ (\underline{e_2} : \mathtt{cons}\ m\ x\ xs \equiv_{\mathtt{Vec}\ A\ e_1} \mathtt{cons}\ n\ y\ ys)\end{array}$$

$$\simeq \begin{array}{l}(\underline{m}\ n : \mathbb{N})(x\ \underline{y} : A)(xs : \mathtt{Vec}\ A\ m)(\underline{ys} : \mathtt{Vec}\ A\ n) \\ (\underline{e_1} : m \equiv_{\mathbb{N}} n)(\underline{e_2} : x \equiv_A y)(\underline{e_3} : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys)\end{array}$$

$$\simeq (n : \mathbb{N})(x : A)(xs : \mathtt{Vec}\ A\ n)$$

Reverse unification rules: example

$$
\begin{array}{ll}
& (\underline{n} : \mathbb{N})(x\ y : A)(xs\ ys : \texttt{Vec}\ A\ n) \\
& (e : \texttt{cons}\ n\ x\ xs \equiv_{\texttt{Vec}\ A\ (\texttt{suc}\ n)} \texttt{cons}\ n\ y\ ys) \\
& (m\ n : \mathbb{N})(x\ y : A)(xs : \texttt{Vec}\ A\ m)(ys : \texttt{Vec}\ A\ n) \\
\simeq & (\underline{e_1} : m \equiv_{\mathbb{N}} n) \\
& (e_2 : \texttt{cons}\ m\ x\ xs \equiv_{\texttt{Vec}\ A\ (\texttt{suc}\ e_1)} \texttt{cons}\ n\ y\ ys) \\
& (m\ n : \mathbb{N})(x\ y : A)(xs : \texttt{Vec}\ A\ m)(ys : \texttt{Vec}\ A\ n) \\
\simeq & (e_1 : \texttt{suc}\ m \equiv_{\mathbb{N}} \texttt{suc}\ n) \\
& (\underline{e_2} : \texttt{cons}\ m\ x\ xs \equiv_{\texttt{Vec}\ A\ e_1} \texttt{cons}\ n\ y\ ys) \\
\simeq & (\underline{m}\ n : \mathbb{N})(x\ \underline{y} : A)(xs : \texttt{Vec}\ A\ m)(\underline{ys} : \texttt{Vec}\ A\ n) \\
& (\underline{e_1} : m \equiv_{\mathbb{N}} n)(\underline{e_2} : x \equiv_A y)(\underline{e_3} : xs \equiv_{\texttt{Vec}\ A\ e_1} ys) \\
\simeq & (n : \mathbb{N})(x : A)(xs : \texttt{Vec}\ A\ n)
\end{array}
$$

Reverse unification rules: problems

- Applicability is limited:
  indices need to be linear patterns
- Hard to implement
- Not clear how to apply injectivity
  for indexed data in reverse

Going beyond the first level

Realization: same problem as for case splitting, only for equations instead of variables

We can solve it in the same way as well: by specialization by unification

Going beyond the first level

Realization: same problem as for case splitting, only for equations instead of variables

We can solve it in the same way as well: by specialization by unification

Going beyond the first level

Realization: same problem as for case splitting, only for equations instead of variables

We can solve it in the same way as well: by specialization by unification

Higher-dimensional unification: example

$$(e : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ \underline{(\mathtt{suc}\ n)}} \mathtt{cons}\ n\ y\ ys)$$

$$\begin{aligned}
& (\underline{e_1} : \mathtt{suc}\ n \equiv_{\mathbb{N}} \mathtt{suc}\ n) \\
\simeq\ & (\underline{e_2} : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ e_1} \mathtt{cons}\ n\ y\ ys) \\
& (f : e_1 \equiv_{\mathtt{suc}\ n \equiv_{\mathbb{N}} \mathtt{suc}\ n} \mathtt{refl})
\end{aligned}$$

$$\begin{aligned}
\simeq\ & (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys) \\
& (\underline{f} : \mathtt{cong}\ \mathtt{suc}\ e_1 \equiv_{\mathtt{suc}\ n \equiv_{\mathbb{N}} \mathtt{suc}\ n} \mathtt{refl})
\end{aligned}$$

$$\begin{aligned}
\simeq\ & (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys) \\
& (\underline{f} : e_1 \equiv_{n \equiv_{\mathbb{N}} n} \mathtt{refl})
\end{aligned}$$

$$\simeq (e_2 : x \equiv_A y)(e_3 : xs \equiv_{\mathtt{Vec}\ A\ n} ys)$$

Higher-dimensional unification: example

$$(e : \texttt{cons}\ n\ x\ xs \equiv_{\texttt{Vec}\ A\ \underline{(\texttt{suc}\ n)}} \texttt{cons}\ n\ y\ ys)$$

$$\begin{aligned}
\simeq\ & (\underline{e_1} : \texttt{suc}\ n \equiv_{\mathbb{N}} \texttt{suc}\ n) \\
& (\underline{e_2} : \texttt{cons}\ n\ x\ xs \equiv_{\texttt{Vec}\ A\ e_1} \texttt{cons}\ n\ y\ ys) \\
& (f : e_1 \equiv_{\texttt{suc}\ n \equiv_{\mathbb{N}} \texttt{suc}\ n} \texttt{refl}) \\
\simeq\ & (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\texttt{Vec}\ A\ e_1} ys) \\
& (\underline{f} : \texttt{cong}\ \texttt{suc}\ e_1 \equiv_{\texttt{suc}\ n \equiv_{\mathbb{N}} \texttt{suc}\ n} \texttt{refl}) \\
\simeq\ & (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\texttt{Vec}\ A\ e_1} ys) \\
& (\underline{f} : e_1 \equiv_{n \equiv_{\mathbb{N}} n} \texttt{refl}) \\
\simeq\ & (e_2 : x \equiv_A y)(e_3 : xs \equiv_{\texttt{Vec}\ A\ n} ys)
\end{aligned}$$

Higher-dimensional unification: example

$$(e : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ \underline{(\mathtt{suc}\ n)}} \mathtt{cons}\ n\ y\ ys)$$

$$\begin{array}{ll} & (\underline{e_1} : \mathtt{suc}\ n \equiv_{\mathbb{N}} \mathtt{suc}\ n) \\ \simeq & (\underline{e_2} : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ e_1} \mathtt{cons}\ n\ y\ ys) \\ & (f : e_1 \equiv_{\mathtt{suc}\ n \equiv_{\mathbb{N}} \mathtt{suc}\ n} \mathtt{refl}) \\ \simeq & (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys) \\ & (\underline{f} : \mathtt{cong}\ \mathtt{suc}\ e_1 \equiv_{\mathtt{suc}\ n \equiv_{\mathbb{N}} \mathtt{suc}\ n} \mathtt{refl}) \\ \simeq & (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys) \\ & (\underline{f} : e_1 \equiv_{n \equiv_{\mathbb{N}} n} \mathtt{refl}) \\ \simeq & (e_2 : x \equiv_A y)(e_3 : xs \equiv_{\mathtt{Vec}\ A\ n} ys) \end{array}$$

Higher-dimensional unification: example

$$
\begin{array}{ll}
& (e : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ \underline{(\mathtt{suc}\ n)}} \mathtt{cons}\ n\ y\ ys) \\
\simeq & \begin{array}{l} (\underline{e_1} : \mathtt{suc}\ n \equiv_{\mathbb{N}} \mathtt{suc}\ n) \\ (\underline{e_2} : \mathtt{cons}\ n\ x\ xs \equiv_{\mathtt{Vec}\ A\ e_1} \mathtt{cons}\ n\ y\ ys) \\ (f : e_1 \equiv_{\mathtt{suc}\ n \equiv_{\mathbb{N}} \mathtt{suc}\ n} \mathtt{refl}) \end{array} \\
\simeq & \begin{array}{l} (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys) \\ (\underline{f} : \mathtt{cong}\ \mathtt{suc}\ e_1 \equiv_{\mathtt{suc}\ n \equiv_{\mathbb{N}} \mathtt{suc}\ n} \mathtt{refl}) \end{array} \\
\simeq & \begin{array}{l} (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\mathtt{Vec}\ A\ e_1} ys) \\ (\underline{f} : e_1 \equiv_{n \equiv_{\mathbb{N}} n} \mathtt{refl}) \end{array} \\
\simeq & (e_2 : x \equiv_A y)(e_3 : xs \equiv_{\mathtt{Vec}\ A\ n} ys)
\end{array}
$$

Representing higher-order problems using first-order syntax

An n-dimensional unification problem consists of

- a telescope Γ of flexible variables
- equation telescopes $\Delta_1, \ldots, \Delta_n$ such that $\vdash \Gamma\Delta_1 \ldots \Delta_n$
- left- and right-hand sides $\bar{u}_1, \bar{v}_1, \ldots \bar{u}_n, \bar{v}_n$ such that $\Gamma\Delta_1 \ldots \Delta_{i-1} \vdash \bar{u}_i, \bar{v}_1 : \Delta_i$

Discussion

Higher-dimensional unification seems easier to implement than reverse rules

But maybe it goes too far?

Alternative: use reflection to implement a case splitting tactic based on unification

Discussion

Higher-dimensional unification seems
easier to implement than reverse rules

But maybe it goes too far?

Alternative: use reflection to implement
a case splitting tactic based on unification